Horse_or_Human

June 21, 2020

[1]:
```
import os
import zipfile
```

[2]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
```

[3]:
```
local_zip = 'F:/Courses/COURSERA/Tensorflow in practice/Course1/Week3/
 ↪horse-or-human.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/Week3/horse-or-human')
local_zip = 'F:/Courses/COURSERA/Tensorflow in practice/Course1/Week3/
 ↪validation-horse-or-human.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/Week3/validation-horse-or-human')
zip_ref.close()
```

[4]:
```
# Direactory with training horse pictures
train_horse_dir = os.path.join('/Week3/horse-or-human/horses')
# Directory with training human pictures
train_human_dir = os.path.join('/Week3/horse-or-human/humans')
# Directory with our training horse pictures
validation_horse_dir = os.path.join('/Week3/validation-horse-or-human/horses')
# Directory with our training human pictures
validation_human_dir = os.path.join('/Week3/validation-horse-or-human/humans')
```

[5]:
```
train_horse_names = os.listdir(train_horse_dir)
print(train_horse_names[:10])

train_human_names = os.listdir(train_human_dir)
print(train_human_names[:10])

validation_horse_hames = os.listdir(validation_horse_dir)
print(validation_horse_hames[:10])
```

```
validation_human_names = os.listdir(validation_human_dir)
print(validation_human_names[:10])
```

```
['horse01-0.png', 'horse01-1.png', 'horse01-2.png', 'horse01-3.png',
'horse01-4.png', 'horse01-5.png', 'horse01-6.png', 'horse01-7.png',
'horse01-8.png', 'horse01-9.png']
['human01-00.png', 'human01-01.png', 'human01-02.png', 'human01-03.png',
'human01-04.png', 'human01-05.png', 'human01-06.png', 'human01-07.png',
'human01-08.png', 'human01-09.png']
['horse1-000.png', 'horse1-105.png', 'horse1-122.png', 'horse1-127.png',
'horse1-170.png', 'horse1-204.png', 'horse1-224.png', 'horse1-241.png',
'horse1-264.png', 'horse1-276.png']
['valhuman01-00.png', 'valhuman01-01.png', 'valhuman01-02.png',
'valhuman01-03.png', 'valhuman01-04.png', 'valhuman01-05.png',
'valhuman01-06.png', 'valhuman01-07.png', 'valhuman01-08.png',
'valhuman01-09.png']
```

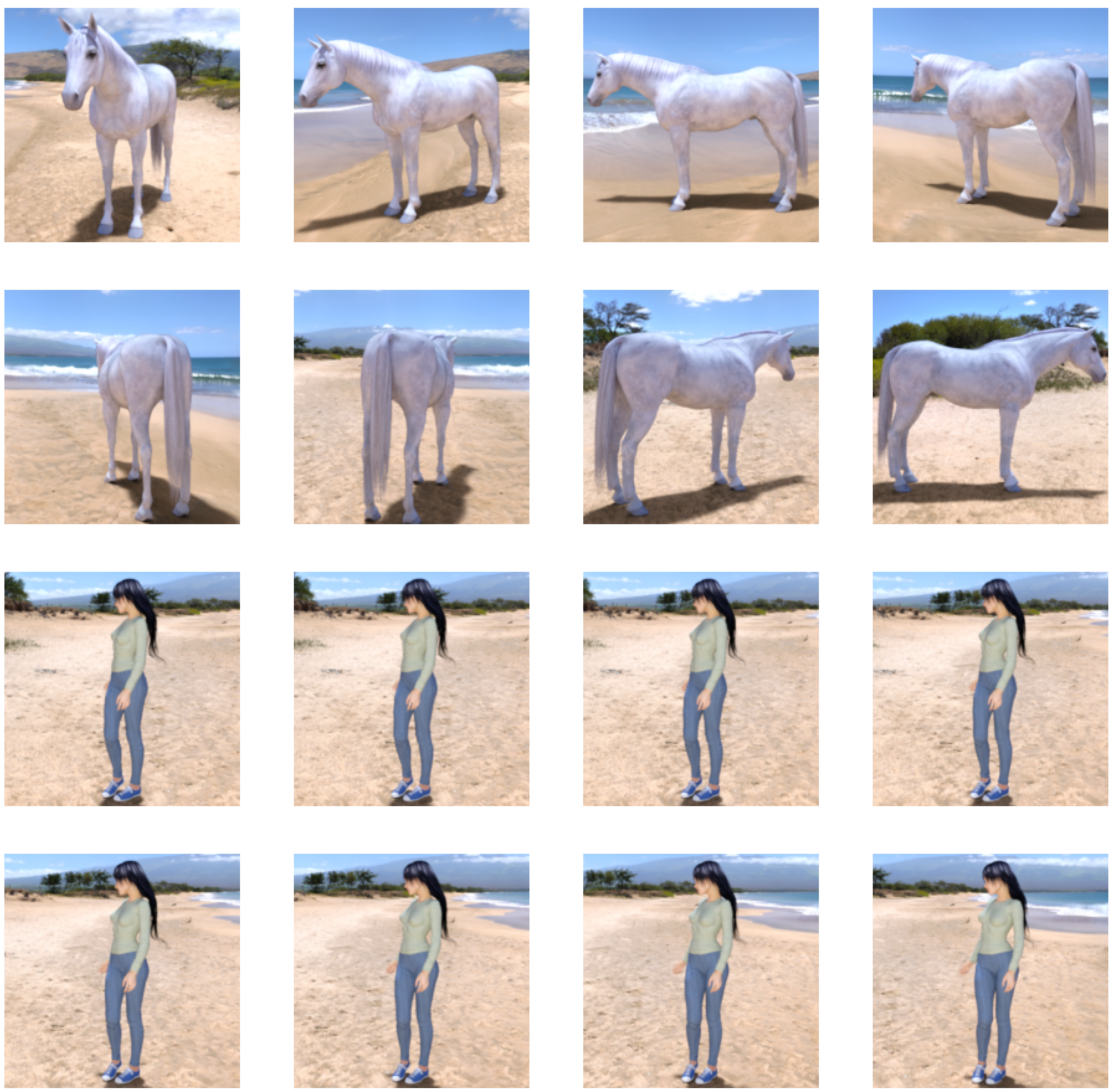
[6]:
```
print('total training horse images:', len(os.listdir(train_horse_dir)))
print('total training human images:', len(os.listdir(train_human_dir)))
print('total validation horse images:', len(os.listdir(validation_horse_dir)))
print('total validation human images:', len(os.listdir(validation_human_dir)))
```

```
total training horse images: 500
total training human images: 527
total validation horse images: 128
total validation human images: 128
```

[7]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

# Index for iterating over images
pic_index = 0
```

[8]:
```
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)

pic_index += 8
next_horse_pix = [os.path.join(train_horse_dir, fname)
                for fname in train_horse_names[pic_index-8:pic_index]]
next_human_pix = [os.path.join(train_human_dir, fname)
```

```
                for fname in train_human_names[pic_index-8:pic_index]]

for i, img_path in enumerate(next_horse_pix+next_human_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```



[9]:
```python
from keras import layers
from keras.models import load_model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
from keras.preprocessing import image
```

```
Using TensorFlow backend.
```

[10]:
```python
model = keras.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation = 'relu', input_shape = (300,␣
 ↪300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid')
])
```

[11]:
```python
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 298, 298, 16)      448

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 149, 149, 16)      0

_________________________________________________________________
conv2d_1 (Conv2D)            (None, 147, 147, 32)      4640

_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 73, 73, 32)        0

_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 64)        18496

_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 35, 35, 64)        0

_________________________________________________________________
conv2d_3 (Conv2D)            (None, 33, 33, 64)        36928

_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
```

```
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1606144
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,704,097
Trainable params: 1,704,097
Non-trainable params: 0
_________________________________________________________________
```

[12]:
```
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

[13]:
```
train_datagen = ImageDataGenerator(rescale = 1 / 255)
validation_datagen = ImageDataGenerator(rescale = 1 / 255)

train_generator = train_datagen.flow_from_directory(
    '/Week3/horse-or-human',
    target_size = (300, 300),
    batch_size = 128,
    class_mode = 'binary'
)

validation_generator = validation_datagen.flow_from_directory(
    '/Week3/validation-horse-or-human',
    target_size = (300, 300),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
Found 1027 images belonging to 2 classes.
Found 256 images belonging to 2 classes.
```

[14]:
```
history = model.fit(
    train_generator,
    steps_per_epoch = 8,
    epochs = 25,
    validation_data = validation_generator,
    validation_steps = 8,
    verbose = 1
```

```
)
```

```
Epoch 1/25
8/8 [==============================] - 77s 10s/step - loss: 0.6999 - accuracy:
0.5106 - val_loss: 0.6343 - val_accuracy: 0.8555
Epoch 2/25
8/8 [==============================] - 57s 7s/step - loss: 0.7825 - accuracy:
0.6974 - val_loss: 1.5911 - val_accuracy: 0.5664
Epoch 3/25
8/8 [==============================] - 54s 7s/step - loss: 0.5067 - accuracy:
0.7754 - val_loss: 0.4540 - val_accuracy: 0.8320
Epoch 4/25
8/8 [==============================] - 48s 6s/step - loss: 1.2750 - accuracy:
0.8376 - val_loss: 0.6009 - val_accuracy: 0.7578
Epoch 5/25
8/8 [==============================] - 48s 6s/step - loss: 0.3579 - accuracy:
0.8888 - val_loss: 0.9362 - val_accuracy: 0.8047
Epoch 6/25
8/8 [==============================] - 48s 6s/step - loss: 0.1836 - accuracy:
0.9321 - val_loss: 1.4812 - val_accuracy: 0.8203
Epoch 7/25
8/8 [==============================] - 49s 6s/step - loss: 0.4523 - accuracy:
0.8454 - val_loss: 1.5145 - val_accuracy: 0.7930
Epoch 8/25
8/8 [==============================] - 48s 6s/step - loss: 0.2172 - accuracy:
0.9266 - val_loss: 1.2552 - val_accuracy: 0.8125
Epoch 9/25
8/8 [==============================] - 48s 6s/step - loss: 0.1064 - accuracy:
0.9611 - val_loss: 1.3325 - val_accuracy: 0.8398
Epoch 10/25
8/8 [==============================] - 55s 7s/step - loss: 0.1177 - accuracy:
0.9521 - val_loss: 1.0588 - val_accuracy: 0.8672
Epoch 11/25
8/8 [==============================] - 55s 7s/step - loss: 0.1046 - accuracy:
0.9622 - val_loss: 0.8720 - val_accuracy: 0.8828
Epoch 12/25
8/8 [==============================] - 49s 6s/step - loss: 0.4172 - accuracy:
0.8754 - val_loss: 4.9993 - val_accuracy: 0.6875
Epoch 13/25
8/8 [==============================] - 48s 6s/step - loss: 0.2082 - accuracy:
0.9511 - val_loss: 1.0767 - val_accuracy: 0.8594
Epoch 14/25
8/8 [==============================] - 48s 6s/step - loss: 0.0470 - accuracy:
0.9833 - val_loss: 1.2796 - val_accuracy: 0.8555
Epoch 15/25
8/8 [==============================] - 47s 6s/step - loss: 0.0211 - accuracy:
0.9922 - val_loss: 1.2359 - val_accuracy: 0.8750
```

```
Epoch 16/25
8/8 [==============================] - 54s 7s/step - loss: 0.0121 - accuracy:
0.9971 - val_loss: 6.2113 - val_accuracy: 0.6445
Epoch 17/25
8/8 [==============================] - 48s 6s/step - loss: 0.6884 - accuracy:
0.8209 - val_loss: 1.7886 - val_accuracy: 0.7617
Epoch 18/25
8/8 [==============================] - 55s 7s/step - loss: 0.0773 - accuracy:
0.9711 - val_loss: 1.1167 - val_accuracy: 0.8672
Epoch 19/25
8/8 [==============================] - 48s 6s/step - loss: 0.0169 - accuracy:
0.9967 - val_loss: 1.3094 - val_accuracy: 0.8750
Epoch 20/25
8/8 [==============================] - 48s 6s/step - loss: 0.0051 - accuracy:
1.0000 - val_loss: 1.7119 - val_accuracy: 0.8555
Epoch 21/25
8/8 [==============================] - 47s 6s/step - loss: 0.0030 - accuracy:
1.0000 - val_loss: 2.4898 - val_accuracy: 0.8164
Epoch 22/25
8/8 [==============================] - 48s 6s/step - loss: 0.5205 - accuracy:
0.8610 - val_loss: 2.0839 - val_accuracy: 0.7500
Epoch 23/25
8/8 [==============================] - 48s 6s/step - loss: 0.0914 - accuracy:
0.9544 - val_loss: 1.6653 - val_accuracy: 0.8242
Epoch 24/25
8/8 [==============================] - 55s 7s/step - loss: 0.0374 - accuracy:
0.9889 - val_loss: 2.1245 - val_accuracy: 0.8203
Epoch 25/25
8/8 [==============================] - 48s 6s/step - loss: 0.0147 - accuracy:
0.9956 - val_loss: 1.5022 - val_accuracy: 0.8789
```

[15]:
```
model.save("horse_or_human.h5")
print("Saved Model to Disk")
```

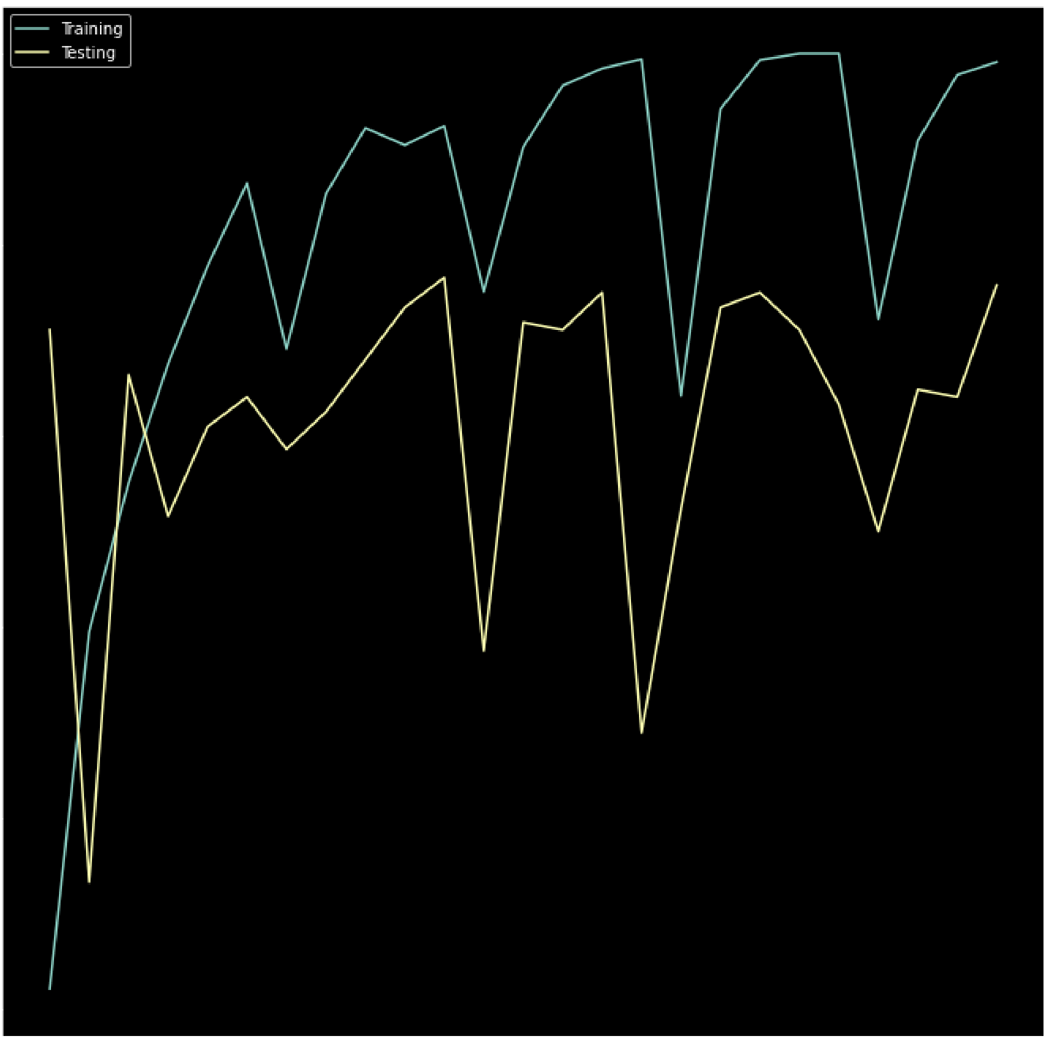
```
Saved Model to Disk
```

[16]:
```
# predicting images
path = 'F:/Courses/COURSERA/Tensorflow in practice/Course1/Week3/human.jpg'
img = image.load_img(path, target_size=(300, 300))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
images = np.vstack([x])
classes = model.predict(images, batch_size=10)
print(classes[0])
if classes[0]>0.5:
    print("It is a human")
else:
```

```
    print("It is a horse")
```

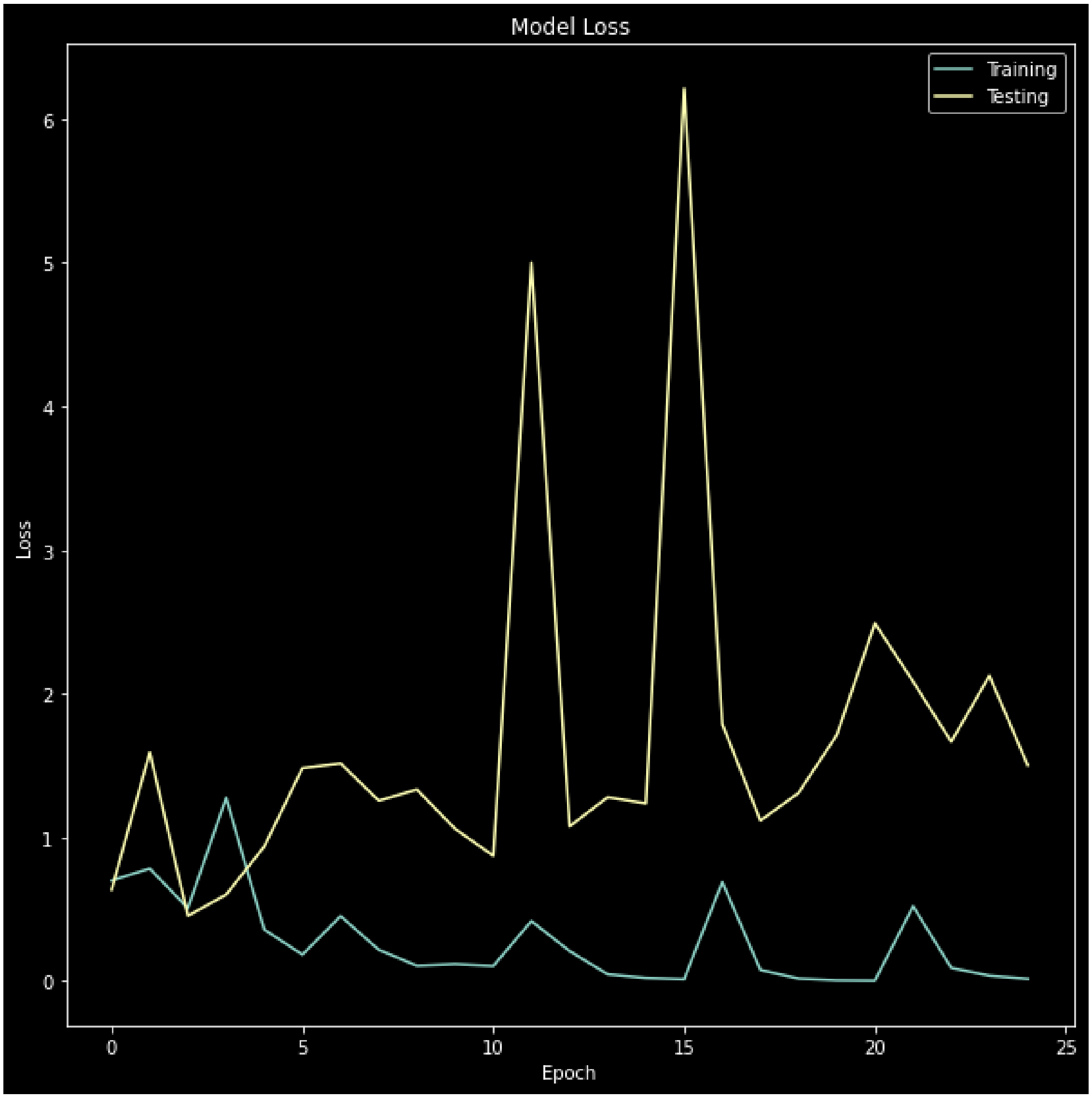
```
[1.]
It is a human
```

[17]:
```
plt.figure(figsize=(10,10))
plt.style.use('dark_background')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Testing'])
plt.tight_layout()
plt.show()
```



[18]:

```
plt.figure(figsize=(10,10))
plt.style.use('dark_background')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Testing'])
plt.show()
```

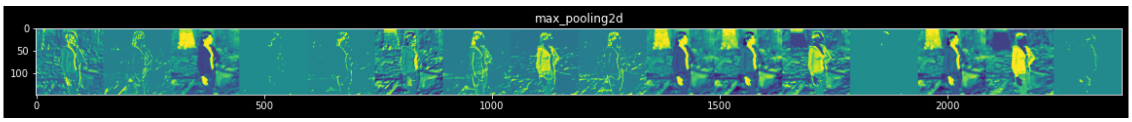


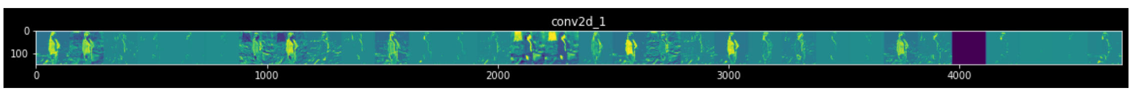
[19]:

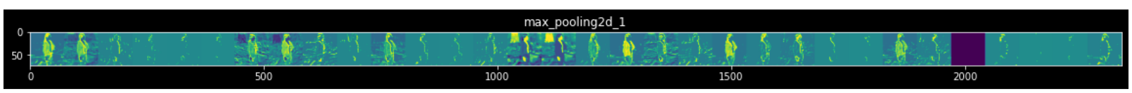
```
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]
#visualization_model = Model(img_input, successive_outputs)
visualization_model = tf.keras.models.Model(inputs = model.input, outputs =␣
↪successive_outputs)
# Let's prepare a random input image from the training set.
horse_img_files = [os.path.join(train_horse_dir, f) for f in train_horse_names]
human_img_files = [os.path.join(train_human_dir, f) for f in train_human_names]
img_path = random.choice(horse_img_files + human_img_files)

img = load_img(img_path, target_size=(300, 300))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255
x /= 255

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers[1:]]

# Now let's display our representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  if len(feature_map.shape) == 4:
    # Just do this for the conv / maxpool layers, not the fully-connected layers
    n_features = feature_map.shape[-1]  # number of features in feature map
    # The feature map has shape (1, size, size, n_features)
    size = feature_map.shape[1]
    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))
    for i in range(n_features):
      # Postprocess the feature to make it visually palatable
      x = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std()
      x *= 64
      x += 128
      x = np.clip(x, 0, 255).astype('uint8')
```

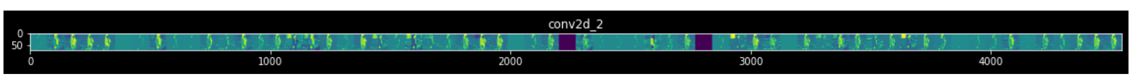
```python
        # We'll tile each filter into this big horizontal grid
        display_grid[:, i * size : (i + 1) * size] = x
    # Display the grid
    scale = 20. / n_features
    plt.figure(figsize=(scale * n_features, scale))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

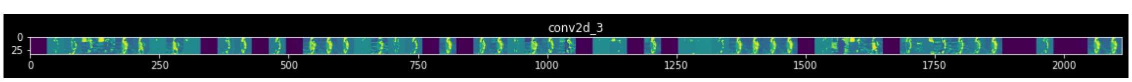
```
<ipython-input-19-d5bfb57b081c>:43: RuntimeWarning: invalid value encountered in
true_divide
  x /= x.std()
```

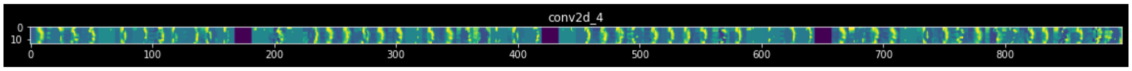


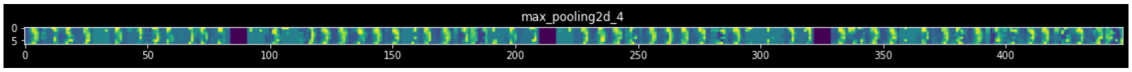









conv2d_4
0
10
0
100
200
300
400
500
600
700
800


max_pooling2d_4
0
5
0
50
100
150
200
250
300
350
400

```
[ ]:
```